Course_Manipulation_Unit_2

September 9, 2023

1 ROS Developers Live Class n57

1.1 Configuring an arm robot to grasp things (Part 1)

This unit will show you how to create a Moveit Package for your industrial robot. By completing this unit, you will be able to create a package that allows your robot to perform motion planning.

1.2 Why this class?

The most interesting thing about having a robot, is making it bring you some stuff. For that, grasping is an essential skill.

Grasping means that the robot is able to identify and grasp an object from a flat surface

In this live class we are going to deal only with the 3 basic steps to grasp the object from a table, provided that:

- **The robot is properly facing the object**
- **There is a perception system that is providing us with the location of the object, realted to the *base_link* frame of the robot**

Pre-requisites for this live class are: * Basic knowledge of ROS concepts such as topics, publish and subscribe, ROS Service, ROS Actions. If you don't know about it, check this course

- Basic knowledge of **ROS TF frames**. If you don't know about it, check this course
- Love for Robotics
- ...that's it!!

1.2.1 How to use this ROSject

A **ROSject** is a **ROS project** packaged in such a way that all the material it contains (**ROS code, Gazebo simulations and Notebooks**) can be shared with any body **using only a web link**. That is what we did with all the attendants to the Live Class, we shared this ROSject with them (so they can have access to all the ROS material they contain).

Check this webinar to learn more about ROSjects and how to create your own ROSjects.

You will need to have a free account at the ROS Development Studio (ROSDS). Get the account and then follow the indications below.

1.3 What is MoveIt?

MoveIt is a ROS framework that allows you to perform motion planning with an specific robot. And... what does this mean? Well, it bascially means that it allows you to plan a movement(motion) from a point A to a point B, without colliding with anything.

MoveIt is a very complex and useful tool. So, within this MicroCourse, we are not going to dive into the details of how MoveIt works, or all the features it provides. If you are intereseted in learning more about MoveIt, you can have a look ath the official website here: http://moveit.ros.org/

Fortunately, MoveIt provides a very nice and easy-to-use GUI, which will help us to be able to interact with the robot in order to perform motion planning. However, before being able to actually use MoveIt, we need to build a package. This package will generate all the configuration and launch files required for using our defined robot (the one that is defined in the URDF file) with MoveIt. In order to generate this package, just follow all the steps described in the following exercise!

1.4 Simulation of today

Today we are going to use the simulation of a grasping system, including an industrial arm and a gripper

To launch it, select **Simulations->from my workspace** then select the **shadown_gazebo/main.launch** file.

1.4.1 Generating MoveIt! configuration package using Setup Assistant tool

Exercise 2.1

a) First of all, you'll need to launch the MoveIt Setup Assistant. You can do that by typing the following command:

Execute in WebShell #1

roslaunch moveit_setup_assistant setup_assistant.launch

This opens a QT window inside the **Graphical Tools** (menu Tools->Graphical Tools)

Great! You now are at the MoveIt Setup Assistant. The next thing you'll need to is to load your robot file. So let's continue!

b) Click on the "Create New MoveIt Configuration Package" button. A new section like this will appear:

Now, just click the "Browse" button, select the URDF file named **model.urdf** located in the **smart_grasping_sandbox/fh_desc** package, and click on the "Load Files" button. You will probably need to copy this file into your workspace. You should now see something like this:

Great! So now, you've loaded the xacro file of your robot to the MoveIt Setup Assistant. Now, let's start configuring some things.

c) Go to the "Self-Collisions" tab, and click on the "Regenerate Default Collision Matrix" button. You will end with something like this:

Here, you are just defining some pairs of links that don't need to be considered when performing collision checking. For instance, because they are adjacent links, so they will always be in collision.

d) Next, move to the "Virtual Joints" tab. Here, you will define a virtual joint for the base of the robot. Click the "Add Virtual Joint" button, and set the name of this joint to FixedBase, and the parent to world. Just like this:

Finally, click the "Save" button. Basically, what you are doing here is to create an "imaginary" joint that will connect the base of your robot with the simulated world.

e) Now, open the "Planning Groups" tab and click the "Add Group" button. Now, you will create a new group called arm, which will use the KDLKinematicsPlugin. Just like this:

Also, select one of the Default Planners for OMPL Planning. For instance, RRT or RRTConnect.

Next, you will click on the "Add Joints" button, and you will select all the joints that form the arm of the robot, excluding the gripper. Just like this:

Finally, click the "Save" button and you will end up with something like this:

So now, you've defined a group of links for performing Motion Planning with, and you've defined the plugin you want to use to calculate those Plans.

Now, repeat the same process, but this time for the gripper. In this case, you DO NOT have to define any Kinematics Solver. If you are not sure of what joints to add to the hand, you can have a look at the below image.

At the end, you should end up with something similar to this:

f) Now, you are going to create a couple of predefined poses for your robot. Go to the "Robot Poses" tab and click on the "Add Pose" button. In the left side of the screen, you will be able to define the name of the pose and the planning group it's refered to. In this case, we will name the 1st Pose open, and it will be related, obviously, to the hand Group.

Now, you will have to define the positions of the joints that will be related to this Pose. For this case, you can set them as in the image below:

Now, repeat the operation, but this time we will define the close Pose. For instance, it could be something like this:

Finally, let's create an start Pose for the arm Group. It could be something like this:

At the end, you should have something similar to this:

g) The next step will be to set up the End-Effector of the robot. For that, just go the End Effectors tab, and click on the "Add End Effector" button. We will name our End Effector hand.

h) Now, just enter your name and e-mail at the "Author Information" tab.

i) Finally, go to the "Configuration Files" tab and click the "Browse" button. Navigate to the catkin_ws/src directory, create a new directory, and name it myrobot_moveit_config. "Choose" the directory you've just created.

Now, click the "Generate Package" button. If everyting goes well, you should see something like this:

And that's it! You have just created a MoveIt package for your articulated robot.

End of Exercise 2.1

Data for Exercise 2.1

Check the following Notes in order to complete the Exercise: Note 1: If, for any reason, you need to edit your MoveIt package (for instances, in futures exercises you detect that you did an error), you can do that by selecting the Edit Existing Moveit Configuration Package option in the Setup Assistant, and then selecting your package. Note 2: If you modify your MoveIt package, you will need to restart the simulation in order to make this changes to have effect.

And that's it! You've created your MoveIt package for your robot. But... now what?

Now that you've already created a MoveIt package, and you've worked a little bit with it, let's take a deeper look at some key aspects of Moveit.

Let's start with a quick look at MoveIt architecture. Understanding the architecture of MoveIt! helps to program and interface the robot to MoveIt. Here, you can have a look at a diagram showing MoveIt architecture.

1.4.2 The move_group node

We can say that **move group** is the heart of MoveIt, as this node acts as an integrator of the various components of the robot and delivers actions/services according to the user's needs.

The **move group** node collects robot information, such as the PointCloud, the joint state of the robot, and the transforms (TFs) of the robot in the form of topics and services.

From the parameter server, it collects the robot kinematics data, such as robot description (URDF), SRDF (Semantic Robot Description Format), and the configuration files. The SRDF file and the configuration files are generated while we generate a MoveIt! package for our robot. The configuration files contain the parameter file for setting joint limits, perception, kinematics, end effector, and so on. These are the files that have been created in the **config** folder of your package.

When MoveIt! gets all of this information about the robot and its configuration, we can say it is properly configured and we can start commanding the robot from the user interfaces. We can either use C++ or Python MoveIt! APIs to command the move group node to perform actions, such as pick/place, IK, or FK, among others. Using the RViz motion planning plugin, we can command the robot from the RViz GUI itself. And this is what you are going to do in the next section!

1.5 Basic Motion Planning

Well... to start, you can just launch the MoveIt Rviz environment and begin to do some tests about Motion Planning. So, follow the next exercise in order to do so!

Exercise 2.2

a) Execute the following command in order to start the MoveIt RViz demo environment.

Execute in WebShell #1

```
[ ]: roslaunch myrobot_moveit_config demo.launch
```

NOTE: It may happen that the Moveit Rviz window appears out of focus. Like this:

If this is your case, just click on the following button that appears on the top-right corner of the screen:

And after that, click again on the RViz screen. Now, your MoveIt Rviz window should appear like this:

Now, you can just double-click on the top coloured part of the window in order to maximize.

If everything goes OK, you will see something like this:

b) Now, move to the Planning tab. Here:

c) Before start Planning anything, it is always a good practice to update the current Start State.

d) At the query section, in the Goal State, you can choose the start option (which one of the Poses you defined in the Previous Exercise) and click on the "Update" button. Your robot scene will be updated with the new position that has been selected.

e) Now, you can click on the "Plan" button at the "Commands" section. The robot will begin to plan a trajectory to reach that point.

f) Finally, if you click on the "Execute" button, the robot will execute that trajectory.

g) Now just play with the new tool! You can repeat this same process some more times. For instance, instead of moving the robot to the start position, you could set a random valid position as goal. You can also try to check and uncheck the different visualization options that appear in the upper "Displays" section.

End of Exercise 2.2

You've now seen how to perform some basic Motion Planning through the MoveIt RViz GUI, and you're a little more familiar with MoveIt. So... let's discuss some interesting points!

1.5.1 MoveIt! planning scene

The term "planning scene" is used to represent the world around the robot and also store the state of the robot itself. The planning scene monitor inside of move_group maintains the planning scene representation. The move_group node consists of another section called the world geometry monitor, which builds the world geometry from the sensors of the robot and from the user input.

The planning scene monitor reads the joint_states topic from the robot, and the sensor information and world geometry from the world geometry monitor. The world scene monitor reads from the occupancy map monitor, which uses 3D perception to build a 3D representation of the environment, called Octomap. The Octomap can be generated from PointClouds, which are handled by a

PointCloud occupancy map update plugin and depth images handled by a depth image occupancy map updater. You will see this part in the next chapter, when we introduce Perception.

1.5.2 MoveIt! kinematics handling

MoveIt! provides a great flexibility for switching the inverse kinematics algorithms using the robot plugins. Users can write their own IK solver as a MoveIt! plugin and switch from the default solver plugin whenever required. The default IK solver in MoveIt! is a numerical jacobian-based solver.

Compared to the analytic solvers, the numerical solver can take time to solve IK. The package called IKFast can be used to generate a C++ code for solving IK using analytical methods, which can be used for different kinds of robot manipulators and perform better if the DOF is less than 6. This C++ code can also be converted into the MoveIt! plugin by using some ROS tools.

Forward kinematics and finding jacobians are already integrated into the MoveIt! RobotState class, so we don't need to use plugins for solving FK.

1.5.3 MoveIt! collision checking

The CollisionWorld object inside MoveIt! is used to find collisions inside a planning scene, which are using the FCL (Flexible Collision Library) package as a backend. MoveIt! supports collision checking for different types of objects, such as meshes; primitive shapes, such as boxes, cylinders, cones, spheres, and such; and Octomap.

The collision checking is one of the computationally expensive tasks during motion planning. To reduce this computation, MoveIt! provides a matrix called ACM (Allowed Collision Matrix). It contains a binary value corresponding to the need to check for collision between two pairs of bodies. If the value of the matrix is 1, it means that the collision of the corresponding pair is not needed. We can set the value as 1 where the bodies are always so far that they would never collide with each other. Optimizing ACM can reduce the total computation needed for collision avoidance. This was done when you were creating the package, if you remember!

1.6 Moving the real robot

Until now, though, you've only moved the robot in the Moveit application. This is very useful because you can do many tests without worrying about any damage. Anyways, the final goal will be always to move the real robot, right?

The MoveIt package you've created is able to provide the necessary ROS services and actions in order to plan and execute trajectories, but it isn't able to pass this trajectories to the real robot. All the Kinematics you've been performing were executed in an internal simulator that MoveIt provides. In order to communicate with the real robot, it will be necessary to do a couple of modifications to the MoveIt package you've created at the beginning of the Chapter.

Obviously, in this Course you don't have a real robot to do this, so you will apply the same but for moving the simulated robot. In order to see what you need to change in your MoveIt package, just follow the next Exercise.

Exercise 2.3

a) First of all, you'll need to create a file to define how you will control the joints of your "real" robot. Inside the config folder of your moveit package, create a new file named controllers.yaml. Copy the following content inside it:

[ ]:
```
controller_list:
  - name: arm_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    joints:
      - shoulder_pan_joint
      - shoulder_lift_joint
      - elbow_joint
      - wrist_1_joint
      - wrist_2_joint
      - wrist_3_joint
  - name: hand_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    joints:
      - H1_F1J1
      - H1_F1J2
      - H1_F1J3
      - H1_F2J1
      - H1_F2J2
      - H1_F2J3
      - H1_F3J1
      - H1_F3J2
      - H1_F3J3
```

So basically, here you are defining the Action Servers that you will use for controlling the joints of your robot.

First, you are setting the name of the joint trajectory controller Action Server for controlling the arm of the robot. And how do you know that? Well, if you do a rostopic list in any Web Shell, you'll find between your topics the following structure:

So this way, you can know that your robot has a joint trajectory controller Action Server that is called /arm_controller/follow_joint_trajectory/.

Also, you can find out by checking the message that this Action uses, that it is of the type FollowJointTrajectory.

Finally, you already know the names of the joints that your robot uses. You saw them while you were creating the MoveIt package, and you can also find them in the model.urdf file, at the fh_desc package.

Then, it simply repeats the process described just now, but for the /hand_controller/follow_joint/trajectory action server.

b) Next, you'll have to create a file to define the names of the joints of your robot. Again inside the config directory, create a new file called joint_names.yaml, and copy the following content

in it:

```
controller_joint_names: [shoulder_pan_joint, shoulder_lift_joint, elbow_joint,␣
↪wrist_1_joint, wrist_2_joint, wrist_3_joint, H1_F1J1, H1_F1J2, H1_F1J3,␣
↪H1_F2J1, H1_F2J2, H1_F2J3, H1_F3J1, H1_F3J2, H1_F3J3]
```

c) Now, if you open the smart_grasping_sandbox_moveit_controller_manager.launch.xml, which is inside the launch directory, you'll see that it's empty. Put the next content inside it:

```
<launch>
  <rosparam file="$(find myrobot_moveit_config)/config/controllers.yaml"/>
  <param name="use_controller_manager" value="false"/>
  <param name="trajectory_execution/execution_duration_monitoring"␣
↪value="false"/>
  <param name="moveit_controller_manager"␣
↪value="moveit_simple_controller_manager/MoveItSimpleControllerManager"/>
</launch>
```

What you are doing here is basically load the controllers.yaml file you just created, and the MoveItSimpleControllerManager plugin, which will allow you to send the plans calculated in MoveIt to your "real" robot, in this case, the simulated robot.

d) Finally, you will have to create a new launch file that sets up all the system to control your robot. So, inside the launch directory, create a new launch file called myrobot_planning_execution.launch.

```
<launch>

  <rosparam command="load" file="$(find myrobot_moveit_config)/config/
↪joint_names.yaml"/>

  <include file="$(find myrobot_moveit_config)/launch/planning_context.launch" >
    <arg name="load_robot_description" value="true" />
  </include>

  <node name="joint_state_publisher" pkg="joint_state_publisher"␣
↪type="joint_state_publisher">
    <param name="/use_gui" value="false"/>
    <rosparam param="/source_list">[/joint_states]</rosparam>
  </node>

  <include file="$(find myrobot_moveit_config)/launch/move_group.launch">
    <arg name="publish_monitored_planning_scene" value="true" />
  </include>

  <include file="$(find myrobot_moveit_config)/launch/moveit_rviz.launch">
    <arg name="config" value="true"/>
  </include>
```

```
</launch>
```

So finally here, we are loading the joint_names.yaml file, and launching some launch files we need in order to set up the MoveIt environment. You can check what those launch file do, if you want. But let's focus a moment on the joint_state_publisher node that is being launched.

If you do again a rostopic list, you will see that there is a topic called /joint_states. Into this topic is where the state of the joints of the simulated robot are published. So, we need to put this topic into the /source_list parameter, so MoveIt can know where the robot is at each moment.

f) Finally, you just have to launch this launch file you just created (my-robot_planning_execution.launch) and Plan a trajectory, just as you learnt to do in the previous exercise. Once the trajectory is planned, you can press the "Execute" button in order to execute the trajectory in the simulated robot.

End of Exercise 2.3

2 Mission completed!!

3 Your homework, should you choose to accept it, ...

- Create the config file for Fetch Robot

3.0.1 Before you log off, remember to GIVE US A LIKE and hit the THUMBS UP and SUBSCRIBE for more weekly tutorials!!!

3.1 Don't miss the most practical online ROS Conference of 2019!!

A **hands on conference** where you will **learn and practice** at the same time **with the speakers**.

It is an online conference with the same format of the Live Classes. You can attend from anywhere and will get a **rosject** with all the content of the speakers.

8 speakers - 8 practical ROS projects on a single weekend

3.1.1 You can also be a speaker of the conference. Check the call for papers

3.1.2 More information here: www.rosdevcon.com

3.1.3 Check the videos of the previous ROSDevCon 2018 here

We have an online academy that teaches you more about how to control robots with ROS.

3.1.4 Check the following related courses with even deeper explanations:

- ROS Navigation in 5 days
- Mastering with ROS: Summit XL

4 KEEP PUSHING YOUR ROS LEARNING!

[ ]: